U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Programação Funcional e em Lógica

Trabalho Prático 1

Realizado por:

Daniel Dória Pinto (up202108808)

Mariana Marujo Conde (up202108824)

Grupo:

Traxit_1

Instalação e execução

Linux:

- Compilar ficheiro menu ([menu]);
- Correr função play (play.).

Windows:

- Compilar ficheiro menu (consult(*path_to_menu*))
- Correr função play (play.).

Descrição do jogo

Traxit é um jogo de tabuleiro abstrato, em que o objetivo é fazer com que as peças estejam o mais próximo possível do centro do tabuleiro. O mesmo pode ser jogado até quatro jogadores, no entanto, esta versão só está disponível para dois.

Esta variação é jogada num tabuleiro 8×8, tendo cada jogador direito a duas peças, duas vermelhas ('R') e duas azuis ('B'):

- É jogado por dois jogadores adversários, que jogam à vez;
- Duas peças não podem ocupar o mesmo espaço, simultaneamente;
- O jogador só pode mover uma peça por jogada;
- O jogador pode mover cada peça de acordo com o caminho que a carta escolhida pelo adversário mostra;
- Cada peça não pode passar por cima de outra peça durante o movimento;
- Não é possível capturar peças do adversário;
- Se não houver movimentos disponíveis para mexer a peça, esta é reposicionada no canto do tabuleiro;
- O jogo termina quando acabarem as cartas que mostram os caminhos que as peças podem percorrer.

As regras mais pormenorizadas do jogo podem ser consultadas no site *Board Game Geek* através do link: https://boardgamegeek.com/boardgame/392652/traxit

Lógica do jogo

Representação interna do estado do jogo:

- **Board**

O estado atual é representado por uma lista de listas, em que cada uma destas linhas corresponde a uma linha do tabuleiro.

Estado inicial:

```
initialBoard([
    [empty,empty,b,empty,empty,b,empty,empty],
    [empty,empty,empty,empty,empty,empty,empty,empty],
    [empty,empty,empty,empty,empty,empty,empty,empty],
    [empty,empty,empty,empty,empty,empty,empty,empty],
    [empty,empty,empty,empty,empty,empty,empty,empty],
    [empty,empty,empty,empty,empty,empty,empty,empty],
    [empty,empty,empty,empty,empty,empty,empty,empty],
    [empty,empty,r,empty,empty,r,empty,empty]]).
```

Exemplo de Estado Intermédio:

```
([
    [empty,empty,empty,empty,empty,empty,empty,empty],
    [empty,empty,empty,empty,empty,empty,empty,empty],
    [empty,empty,empty,empty,empty,r,empty,empty],
    [empty,empty,empty,empty,empty,empty,empty,empty],
    [empty,empty,r,empty,b,empty,empty,empty],
    [empty,empty,empty,empty,empty,empty,empty,empty],
    [empty,b,empty,empty,empty,empty,empty,empty],
    [empty,empty,empty,empty,empty,empty,empty,empty]]).
```

Exemplo de Estado Final:

([

[empty,empty,empty,empty,empty,empty,empty,empty],

[empty,empty,empty,empty,empty,empty,empty,empty],

[empty,empty,empty,empty,empty,empty,empty,empty],

[empty,empty,empty,r,b,empty,empty,empty],

[empty,empty,empty,empty,b,empty,empty,empty],

[empty,empty,r,empty,empty,empty,empty,empty],

[empty,empty,empty,empty,empty,empty,empty,empty],

[empty,empty,empty,empty,empty,empty,empty,empty]]).

Legenda:

r – peça vermelha (R)

b – peça azul (B)

empty – espaço em branco

Sempre que um jogador faz uma jogada, é criado um novo tabuleiro com as mesmas características do instanciado no início, no entanto a disposição das peças é atualizada para a pretendida pelo Player.

A gestão de quem deve jogar e das cartas ainda disponíveis, bem como o número de rondas, ao longo do jogo, é feita por variáveis no loop principal do jogo:

```
game_loop(Player, Board, BlueCards, RedCards, Round)
```

- **Player**

É o jogador da ronda atual, RedCards e BlueCards correspondem às cartas ainda disponíveis para serem usadas durante a partida.

Visualização do estado de jogo:

Sempre que o jogo volta ao primeiro jogador (azul), uma nova ronda começa. Então no início de cada iteração do mesmo é chamado o predicado display_current_round(Round) para que os jogadores fiquem a par da jogada em que estão.

Para além disto, é necessário especificar qual jogador é que está a jogar, usando display_players_turn(Player), dá-se print a uma mensagem que identifica de quem é o turno em questão.

Visto que o estado atual do nosso jogo é representado pelo tabuleiro, no início de todas as jogadas é chamado o predicado display_board(Board).

Esta função faz um ciclo em que chama os predicados print_board(List, RowNum) e print_row(List) como auxiliares para desenhar o estado atual do Board.

O display_board(Board) num primeiro momento dá print às colunas do tabuleiro e à primeira linha de separação de células. De seguida chama o print_board(List, RowNum) que está responsável por imprimir linhas do tabuleiro e respectivos números, com o auxílio do print_row(List).

Devido às especificações do nosso jogo, o nosso Board não pode ter um tamanho variável uma vez que depende do posicionamento dos peões para serem contados os pontos de cada jogador.

Execução de Jogadas:

A execução da jogada começa com o estado atual do tabuleiro a ser mostrado ao utilizador para este poder escolher que jogada quer fazer.

A escolha do movimento dá-se de seguida, sendo que para a execução de uma jogada é necessário o jogador escolher uma peça (seleciona através da coluna e fila).

```
check_starting_position(Board, Column, Row, Player, ReturnRow. ReturnColumn) :-
    nl, write('Choose a piece to move: '), nl,
    write('\nColumn'),
    checkColumn(ValidColumn, InputColumn),
    write('\nRow: '),
    checkRow(ValidRow, InputRow),
    nth1(InputRow, Board, RowList),
    nth1(InputColumn, RowList, Piece),
    (
```

```
        (Piece == 'r' ; Piece == 'b') ->
        (
            Piece == 'r' ->
                  Pieces = 1
            ;     Pieces = 2),
            (Pieces == Player) ->
            (
                 ReturnColumn is InputColumn,
                 ReturnRow is InputRow,
                 write('\nYour turn to make a move!\n'),
            )
            ;
        write('\nNot your piece! That piece belongs to the other player!\n'),
        write('\nTry again!\n'),
        check_starting_position(Board, Column, Row, Player)
    )
    ;
    write('\nNo piece in that position!\n'),
    check_starting_position(Board, Column, Row, Player)
).
```

São feitas validações para a peça que o jogador escolhe, de forma a garantir que a peça existe e que lhe pertence, e para saber para onde o mesmo a vai mover vai ter de se escolher uma carta com um caminho, então usa-se:

```
display_card_numbers(Cards) :-
      sleep(0.8),
      write('Your cards: '),
      display_card_numbers_aux(Cards).

display_card_number_aux([]) :- nl.
display_card_numbers_aux([[Number, _] | Rest]):-
      display_card(Number),
      sleep(0.4),
      display_card_numbers_aux(Rest).
```

Para verificação da validade de uma jogada, usamos o predicado sum_coords(Card, ReturnRow, ReturnColumn, List, Board), onde o argumento Board será o tabuleiro atual e a forma como as peças estão dispostas no mesmo aquando da verificação da validade da jogada em questão. ReturnRow e ReturnColumn indicam a futura posição da peça, caso o movimento seja validado. Para isto, é necessário que ambos os valores estejam entre 1 e 8 e que a casa do tabuleiro para onde a peça vai esteja *empty*. Para verificar se a casa está vazia chama-se o predicado auxiliar isEmpty(Board, Row, Column) que retorna um booleano dependendo se a casa está ou não vazia.

Depois disto, se a jogada for inválida, novas informações são pedidas ao jogador, e por outro lado, se a jogada for válida, o movimento é executado.

Após este processo, são então apresentadas todas as coordenadas possíveis para onde a peça pode ir, tendo em conta as coordenadas do tabuleiro. Assim o jogador do respetivo turno pode decidir para onde vai a sua peça, tendo em conta a seleção da carta feita pelo jogador adversário.

Após o movimento ser feito, dá-se update à lista de cartas disponiveis do jogador usando o predicado remove_card(Id, Cards, RemainingCards).

```
remove_card(_, [], []).
remove_card(Id, [[Id, _] | Tail], RemainingCards) :-
      !, remove_card(Id, Tail, RemainingCards).
remove_card(Id, [Card | Tail], [Card | RemainingCards]) :-
      remove_card(Id, Tail, RemainingCards).
```

Depois de removida a carta, atualiza-se o board com as novas coordenadas da peça do utilizador chamando dois predicados:

```
remove_piece(Board, Row, Col, NewBoard) :-
    nth1(Row, Board, BoardRow),
    replace(BoardRow, Col, empty, NewRow),
    replace(Board, Row, NewRow, NewBoard).


change_cell(Board, Row, Col, NewValue, NewBoard) :-
    nth1(Row, Board, OldRow),
    replace(Old, Col, NewValue, ModifiedRow),
    replace(Board, Row, ModifiedRow, NewBoard).
```

Para isto foi também necessário criar o predicado replace.

```
replace([_|T], 1, X, [X|T]).
replace([H|T], I, X, [H|R]):-
    I > 1,
    I1 is I - 1,
    replace(T, I1, X, R).
```

Para finalizar o ciclo, troca-se o id do Player e, se se trocar do jogador azul para o vermelho, soma-se 1 ao número da ronda e volta-se a correr o mesmo ciclo com as variáveis atualizadas.

Final do Jogo:

Para o jogo acabar, uma vez que não há remoção de peças do tabuleiro nem capturas, basta apenas chegar à ronda limite que, neste caso, se trata da ronda 12. O predicado que deteta se de facto o jogo terminou é:

```
game_loop(Board,_,7,_,_) :-
  display_board(Board),
  display_winner(Winner).
```

```
display_winner(Winner) :-
  (
        Winner == 1
  ->    print('_____________________________________'),nl,
        print('|                                   |'),nl,
        write('|       RED PLAYER WINS!            |'),nl,
        print('|___________________________________|'),nl,
  ;     Winner == 2
  ->    print('_____________________________________'),nl,
        print('|                                   |'),nl,
        write('|      BLUE PLAYER WINS!            |'),nl,
        print('|___________________________________|'),nl,
  ;     Winner == 3
  ->    print('_____________________________________'),nl,
        print('|                                   |'),nl,
        write('|          ITS A DRAW               |'),nl,
        print('|___________________________________|'),nl,
  ).
```

Lista de Jogadas Válidas:

Existem três principais razões para uma jogada ser válida. A peça escolhida ser válida, a carta escolhida ser válida e as coordenadas finais da peça serem válidas.

Para a peça escolhida ser válida, verifica-se os inputs do user com o predicado check_starting_position(Board, Column, Row, Player, ReturnRow, ReturnColumn), onde vai verificar se de facto as coordenadas submetidas correspondem a uma peça e se esta faz parte do jogador em questão.

Para a carta escolhida ser válida, através dos predicados is_valid_card_number(Id, Cards) e get_card_by_id(Id, Cards, Card) é possível calcular para onde a peça escolhida anteriormente pelo jogador vai parar.

Para verificar se as coordenadas do destino da peça são válidas, verificou-se através dos predicados sum_coords(Card, ReturnRow, ReturnColumn, List, Board) e isEmpty(Board, Row, Column) se as coordenadas estavam dentro dos limites do tabuleiro e se as mesmas já não estavam ocupadas por outras peças, respetivamente.

Conclusões

Este projeto, apesar de bastante grande e complexo para uma nova linguagem, foi bastante útil para perceber como programas feitos em outros paradigmas de programação podem ser igualmente conseguidos com programação em lógica. Foi também possível perceber algumas vantagens do paradigma de programação em lógica face aos outros, principalmente no que toca a algoritmos, que necessitam de instruções lógicas complexas.

A maior dificuldade encontrada foi a adaptação ao tipo da linguagem e transformar a teoria na prática, tendo se tornado bastante difícil implementar os conteúdos da maneira pretendida, nomeadamente coisas que em outras linguagens seriam bastante simples de fazer, como remover elementos de uma mesma lista e ela dar update a si mesma.

Uma outra dificuldade encontrada foi a implementação de uma inteligência artificial que conseguisse jogar o jogo, uma vez que eram necessários bastantes inputs, todos eles muito restritos a opções dadas por listas e criar listas que se adaptavam aos mesmos.

Para além disso, não foi possível integrar no projeto todas as características que queríamos, como foi o caso dos níveis de IA, que por falta de conhecimento e de tempo foi impossível de conseguir realizar.

Ainda assim, consideramos o resultado final como satisfatório.

Bibliografia

As regras do jogo e o seu funcionamento foram consultados através dos seguintes sites:

- https://boardgamegeek.com/boardgame/392652/traxit
- https://www.youtube.com/watch?v=icDGTU7UdBw&ab_channel=Danspil